CycleGen: Closing the Loop in Long-Form Text Generation via Loop-Aware Refinement

Anonymous ACL submission

Abstract

Long-Form text generation is critical for applications ranging from creative writing to technical documentation, yet large language models (LLMs) struggle to maintain structural integrity and logical coherence over extended outputs. Existing linear generation paradigms inadequately capture cyclic semantic dependencies, in which subsequent content retroactively constrains preceding segments. To address this, we propose **CycleGen**, a framework that reformulates long-form text generation as a graph-based cyclic synthesis process. CycleGen comprises: (1) a data engine that constructs dependency graphs, applies a MinFAS-inspired algorithm to resolve logical cycles, and encodes backward constraints via closed-loop correction; and (2) a progressive alignment strategy that decouples optimization objectives through multi-stage training with Graph-Critical IPO, DPO, and SimPO. We further introduce **CycleBench** for evaluating long-range cyclic reasoning capabilities. Experiments on WritingBench, LongGenBench, and CycleBench demonstrate that CycleGen outperforms similarly sized models and achieves performance rivaling significantly larger models. We have released our code and CycleBench at the anonymous repository: **CycleGen Repo**.

1 Introduction

Recent years have witnessed significant advances in long-context understanding, with modern LLMs capable of processing contexts up to 1M tokens (Su et al., 2024). However, long-form text generation presents distinct challenges that remain inadequately addressed (Liu et al., 2024; Anil et al., 2022). In practical applications, users often impose multi-dimensional constraints such as prescribed length, structural outlines, and stylistic consistency. Existing models struggle to maintain logical coherence and structural integrity over extended outputs (Guan et al., 2021). As shown in Figure 1

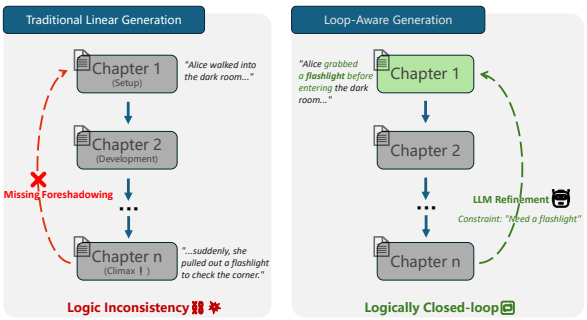

Figure 1: Comparison between unidirectional linear generation and CycleGen's refinement-based cyclic generation. (Left) Traditional models fail to anticipate backward constraints, leading to logical inconsistencies. (Right) CycleGen models cyclic dependencies and uses refinement to achieve a global logical closed loop and successful payoff of foreshadowing.

(Left), traditional linear generation is constrained by unidirectionality and cannot anticipate backward constraints from future content, which often leads to abrupt logical inconsistencies. As generation length increases, models exhibit severe degradation, manifesting as contradictions, discourse breaks, and catastrophic forgetting (Bai et al., 2024; Becker et al., 2024). These issues stem largely from the inability to model cyclic semantic dependencies, where subsequent content retroactively constrains preceding segments. For instance, in a mystery novel, a revelation in Chapter B must align with foreshadowing in Chapter A, forming bidirectional dependencies that linear generation cannot adequately capture.

Prior work has explored this bottleneck from different perspectives. LongWriter (Bai et al., 2024) attributes the problem primarily to distributional bias in supervised fine-tuning data, where the vast majority of training samples are shorter than 2k tokens, preventing models from acquiring long-range generation behaviors. To address generation quality, approaches such as CogWriter (Wan et al.,

2025) and StoryWriter (Xia et al., 2025) adopt plan-then-write paradigms, decomposing long-form text generation into hierarchical sub-tasks via explicit outlines. However, these methods remain fundamentally grounded in linear generation topologies and fail to capture cyclic semantic dependencies.

Bridging the gap between current model capabilities and high-quality long-form text generation requires addressing three core challenges: (1) Modeling cyclic semantic dependencies. Human writing involves non-linear semantic planning with backtracking, where future content retroactively constrains earlier passages. Existing paradigms must forcibly linearize such cyclic structures into left-to-right sequences, systematically discarding backward constraints that are essential for maintaining global coherence. (2) Internalizing explicit planning into implicit generation. Explicit reasoning methods like Chain-of-Thought (Wei et al., 2022) and Tree-of-Thought (Yao et al., 2023) exhibit limited stability in ultra-long contexts. The key challenge is whether models can internalize the explicit process of planning-backtracking-revision into implicit parameter-space capabilities, enabling natural satisfaction of long-range constraints within a single forward pass. (3) Balancing multiple optimization objectives. Long-Form text generation requires simultaneously satisfying length control, logical consistency, and narrative quality. Naive end-to-end reinforcement learning often drives models toward single-dimension extrema, such as maximizing length through redundancy at the expense of overall quality.

To address these challenges, we propose CycleGen, a framework that reformulates long-form text generation as a graph-based cyclic synthesis process.As illustrated in Figure 1 (Right), by explicitly modeling cyclic dependencies, CycleGen employs a refinement mechanism to retroactively revise earlier content based on constraints revealed later, thereby achieving a global logical closed loop with successful payoff of foreshadowing. CycleGen consists of two core components: (1) Data Engine. We construct directed dependency graphs that explicitly allow cycles to represent inter-chapter relationships. We formulate the decycling process as a Minimum Feedback Arc Set (MinFAS) problem and apply a greedy approximation algorithm to selectively remove cyclic edges while converting the graph into a directed acyclic graph (DAG). For tasks with complex narrative logic, we introduce a Loop-Aware Refinement mechanism that reincorporates removed cyclic edges via bidirectional rewriting, repairing logical breaks and ensuring global self-consistency. For hierarchical tasks like analytical reports, the framework naturally degenerates into efficient linear generation. (2) Progressive Alignment Strategy. To tackle the challenges of controllable long-form text generation, we design a multi-phase training pipeline. We first inject long-form writing capabilities via supervised fine-tuning. We then apply a three-stage progressive reinforcement learning scheme (Graph-Critical IPO, DPO, and SimPO) that gradually transitions from basic length control to complex joint structure-logic constraints, internalizing graph-planning capabilities into one-pass generation and mitigating error accumulation from multi-objective optimization.

Our main contributions are summarized as follows:

- We propose CycleGen, a graph-based framework that models cyclic semantic dependencies via MinFAS decoupling and loop aware refinement, overcoming the limitations of linear generation paradigms.
- We design a progressive multi-stage alignment strategy that decouples optimization objectives for logical coherence, length control, and narrative quality, distilling explicit graph-planning into one-pass generation capabilities.
- We introduce CycleBench, a benchmark for evaluating long-range cyclic reasoning. Experiments show that CycleGen outperforms similarly sized models on WritingBench, LongGenBench, and CycleBench, achieving performance rivaling significantly larger models.

2 Related Work

2.1 Long-Form Text Generation Bottlenecks

Despite recent progress in scaling context windows to millions of tokens, LLMs continue to struggle with long-form text generation. LongWriter (Bai et al., 2024) identifies the core bottleneck as distributional bias in supervised fine-tuning data: the scarcity of ultra-long training samples (with the majority under 2K tokens) prevents models from acquiring semantic consistency and structural control across extended outputs. While architectural innovations like RoPE (Su et al., 2024), FlashAttention (Dao et al., 2022), and Mamba (Gu and

Dao, 2024) enable efficient processing of long contexts, they do not directly address generation quality. Building on this data bottleneck hypothesis, we synthesize high-quality long-form training data with explicit cyclic dependencies to better support generation capabilities.

2.2 Controllable Text Generation

Controllable text generation (CTG) aims to produce text satisfying specific attributes such as length, sentiment, or structure. Existing approaches include learning-free methods like FreeCtrl (Feng et al., 2024), which adjusts FFN weights to control keywords, and prompt-based methods like Plan-and-Write (Akinfaderin et al., 2025) for length control. MARKERGEN (Yuan et al., 2025) adopts plug-and-play strategies with external tools, while LDPE (Butcher et al., 2025) injects positional information to internalize length constraints. While effective for short texts or specific tasks, these methods remain limited for long-form text generation: learning-free approaches focus on surface-level constraints and cannot control complex narrative structures, while prompt-based methods rely on linear planning and fail to model cyclic dependencies such as foreshadowing and payoff. In contrast, CycleGen internalizes controllability by constructing training data with explicit non-linear logical dependencies and applying progressive alignment, thereby enabling joint control over length and deep logical consistency.

2.3 Planning-Based Long-Form Text Generation

To address logical discontinuities in long-form text generation, the plan-then-generate paradigm has emerged as an alternative to pure autoregressive decoding. Early hierarchical methods like Re3 (Yang et al., 2022) and DOC (Yang et al., 2023) produce high-level outlines followed by expansion, while recent agent-based approaches like StoryWriter (Xia et al., 2025) and SuperWriter (Wu et al., 2025a) employ multi-agent collaboration and Monte Carlo Tree Search to simulate human writing cycles. Despite improving local coherence, these methods remain constrained by linear or tree-structured topologies that assume unidirectional information flow and cannot model bidirectional dependencies, such as plot twists retroactively constraining earlier descriptions. CycleGen addresses this by modeling plans as directed graphs that explicitly permit cycles and applying Loop-Aware Refinement to handle non-linear logical revisions, achieving global self-consistency.

3 Problem Formulation

3.1 Problem Definition

We formulate the task of long-form text generation as follows: given instruction I, generate an ordered sequence of N semantic blocks (e.g., chapters or key events).

$$\mathcal{S} = \{S_1, S_2, \ldots, S_N\}, \tag{1}$$

with total length $L \gg 10$k tokens. Standard LLMs model this via autoregressive decomposition.

$$P(\mathcal{S} \mid I) \approx \prod_{t=1}^{N} P_\theta(S_t \mid S_{<t}, I). \tag{2}$$

This assumes each block S_t depends only on historical context $S_{<t}$. While this assumption works well for short texts, it fails in long-form narratives with complex non-local dependencies. Purely forward generation then produces outputs that are locally coherent but globally inconsistent.

3.2 Cyclic Semantic Dependencies

To characterize dependencies in long-form text, we distinguish two types of relationships between semantic blocks S_i and S_j ($i < j$):

- **Forward Causal Dependency**: $P(S_j \mid S_i) \neq P(S_j)$. This follows temporal order, where preceding conditions determine subsequent outcomes (e.g., "obtaining a key" enables "opening a chest"). Autoregressive models naturally capture such dependencies.
- **Backward Teleological Constraint**: $P(S_i \mid S_j) \neq P(S_i)$. This reflects authorial intent, where a desired ending S_j retroactively constrains earlier content S_i. For instance, if S_j introduces an unexpected twist, earlier blocks must include appropriate foreshadowing to maintain consistency.

During autoregressive generation of S_i, future blocks S_j remain unobserved, acting as latent variables. The model cannot evaluate $P(S_i \mid S_j, S_{<i})$ and must rely on $P(S_i \mid S_{<i})$. This invisibility of future information limits global planning: without backward constraints, early decisions risk logical conflicts with future developments, causing narrative inconsistency.

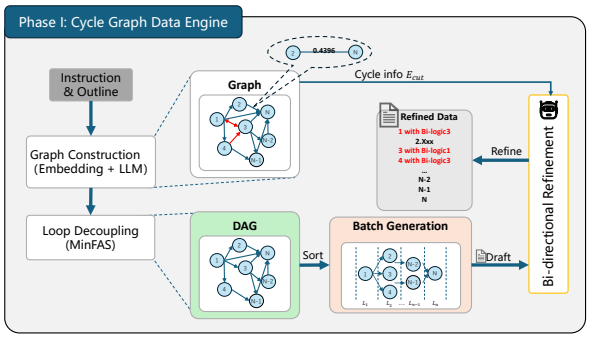

Figure 2: Workflow of the CycleGen data engine (Phase I), illustrating cycle decoupling for linear generation and cycle recovery via bi-directional refinement.

4 Methodology

To address the structural mismatch between linear autoregressive generation and cyclic logical dependencies in long-form text, CycleGen introduces a graph-based closed-loop data engine in Phase I. The engine explicitly models cross-section dependencies and emulates the human writing workflow of planning, drafting, and revision through cycle decoupling and recovery.

4.1 Phase I: Cycle Graph Data Engine

As shown in Figure 2, Phase I aims to construct high-quality training data by explicitly manipulating graph structures to resolve logical consistency issues in long texts. This process involves three steps: graph construction, cycle decoupling, and bi-directional refinement.

Semantic Dependency Graph Construction and Decoupling. Given an instruction and an outline $\mathcal{O} = \{v_1, \ldots, v_N\}$, we first construct a directed weighted graph G to model the logical dependencies between sections. To balance efficiency and accuracy, we adopt a hybrid "retrieval-reasoning" strategy: we first compute the pairwise cosine similarities s_{ij} between nodes using a pre-trained embedding model (In our experiment, we employ Qwen3-Embedding-4B) to filter candidates ($s_{ij} > 0.4$); LLM then infers the causal direction between nodes. (e.g., from "foreshadowing" to "revelation").

Since autoregressive generation requires a topological order, we must convert the cyclic graph G into a DAG. We model this as the Minimum Weight Feedback Arc Set problem, identifying and removing the minimal edge set E_{cut} by weight:

$$E_{cut} = \arg\min_{E' \subset E} \sum_{(u,v) \in E'} w_{uv}, \quad \text{s.t. } G' = (V, E \setminus E') \text{ is a DAG.} \quad (3)$$

The resulting G_{DAG}, after removing E_{cut}, establishes the topological order for section generation, enabling the model to parallelly generate an initial draft $\mathcal{Y}_{raw}$ based on summaries of predecessors and optional RAG retrieval.

Bi-Directional Refinement and Data Synthesis. The initial draft $\mathcal{Y}_{raw}$ inevitably contains logical discontinuities due to the removal of strong dependency edges (E_{cut}). As depicted on the right side of Figure 2, we exploit the cyclic information in the original graph G to perform bi-directional refinement. For each removed dependency $(v_j \to v_i) \in E_{cut}$, we jointly input the drafts y_i and y_j into the refinement model. The model restores logical coherence by either amending y_i (e.g., adding a missing prerequisite) or revising y_j (e.g., adjusting a subsequent development), ultimately generating a logically self-consistent text $\mathcal{Y}_{refined}$. This process not only produces high-quality text but also naturally constructs the preference data pairs $(\mathcal{Y}_{refined}, \mathcal{Y}_{raw})$ for the subsequent alignment stage.

4.2 Phase II: Progressive Alignment Strategy

Long-Form text generation involves multiple, conflicting objectives: micro-level logical consistency, macro-level length compliance, and overall narrative quality. Directly training on synthesized data is often insufficient to resolve these multi-objective conflicts. Naive end-to-end optimization may converge to suboptimal trade-offs, such as satisfying length constraints at the expense of coherence via redundant content.

To address this, CycleGen adopts a progressive alignment strategy, decomposing alignment into staged objectives that distill explicit planning and revision behaviors into the model's implicit generation capability. As shown in Figure 3, this staged pipeline sequentially addresses different constraint dimensions.

Stage 0: Supervised Fine-Tuning (SFT). We first perform supervised fine-tuning on high-quality corpora $\mathcal{D}_{sft}$ synthesized by the data engine, enabling the model to adapt to long-context attention

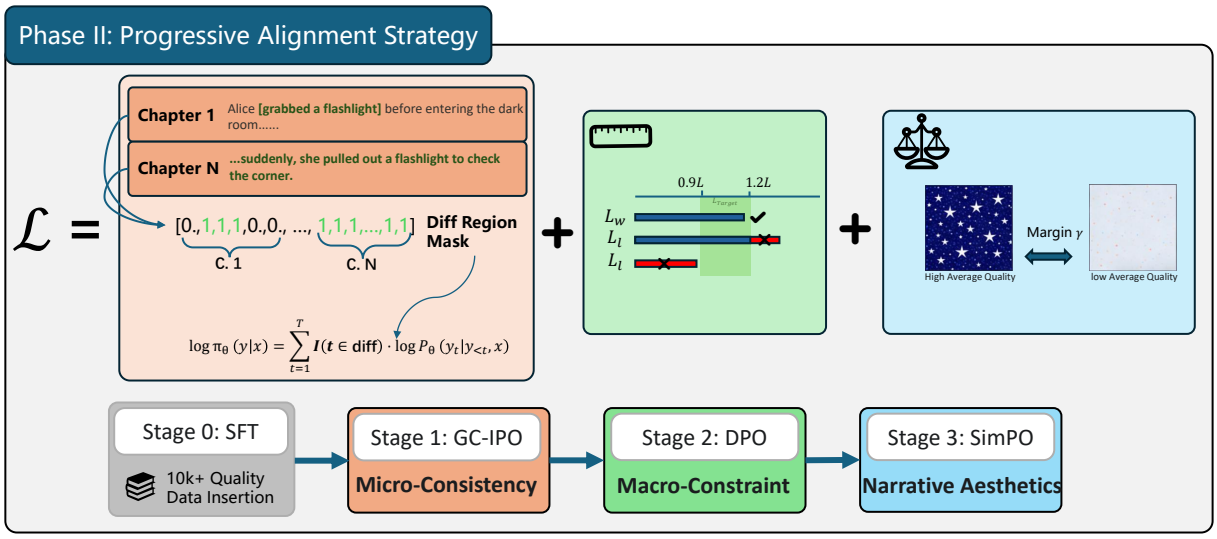

Figure 3: Architecture of the Progressive Alignment Strategy (Phase II). To balance multiple constraints in long-form text generation, we design a staged pipeline: Stage 1 (GC-IPO) focuses on local logical repairs; Stage 2 (DPO) enforces global length interval constraints; and Stage 3 (SimPO) optimizes narrative aesthetics based on length normalization.

patterns and basic formatting conventions:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{sft}} \sum_{t=1}^{|y|} \log P_\theta(y_t \mid y_{<t}, x). \tag{4}$$

Stage 1: Graph-Critical IPO for Logical Consistency. This stage targets long-range logical dependencies using preference pairs (y_w, y_l), where y_w is refined via loop-aware correction and y_l is the original linear draft. A primary challenge is Gradient Dilution. Since y_w and y_l often share over 90% token overlap, direct sequence-level likelihood comparison risks drowning critical correction signals in gradients from unchanged tokens. We therefore propose Graph-Critical IPO. Using graph structure to localize difference regions, we define a masked log-probability:

$$\log \pi_\theta(y|x) = \sum_{t=1}^{T} w_t \cdot \log P_\theta(y_t|y_{<t}, x), \tag{5}$$

where $w_t = 1$ for tokens in critical difference regions and their neighborhoods, and $w_t = \lambda$ otherwise ($\lambda = 0.1$ in practice). This localized weighting focuses optimization on logical bifurcation points. We adopt the IPO objective (Gheshlaghi Azar et al., 2024):

$$\mathcal{L}_{\text{GC-IPO}} = \mathbb{E}_{\mathcal{D}} \left[\left(\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \frac{\tau}{2} \right)^2 \right], \tag{6}$$

encouraging the model to internalize explicit revision behavior as implicit logical intuition.

Stage 2: DPO for Strict Length Compliance. Having established logical coherence, Stage 2 enforces strict adherence to a target length L_{target}. Preference data $\mathcal{D}_{len}$ are constructed based on specific length criteria: y_w must satisfy $|y_w| \in [0.9L_{target}, 1.2L_{target}]$, while y_l falls outside this range ($|y_l| < 0.9L_{target}$ or $|y_l| > 1.2L_{target}$). We apply standard DPO loss over full sequences, enabling the model to control length via pacing rather than truncation or padding. Formally, we adopt the standard DPO objective (Rafailov et al., 2023):

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_{len}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right] \tag{7}$$

Stage 3: SimPO for Structure–Quality Joint Optimization. The final stage addresses the trade-off between length and overall generation quality. Since summed log-probabilities introduce length bias favoring longer outputs, we adopt SimPO (Meng et al., 2024) with length normalization and a target margin γ:

$$\mathcal{L}_{\text{SimPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\frac{\beta}{|y_w|}\log\pi_\theta(y_w|x) - \frac{\beta}{|y_l|}\log\pi_\theta(y_l|x) - \gamma\right)\right]. \quad (8)$$

This formulation encourages higher average generation quality per token, producing texts that are both long and structurally well-formed.

Through these progressive stages, CycleGen disentangles multi-dimensional constraints, resulting in a one-pass long-from text generator that simultaneously achieves logical closure, strict length adherence, and high narrative quality.

5 Experiments

We evaluate the proposed CycleGen through various experiments to demonstrate its effectiveness. Before presenting the experimental results, we first describe the experimental setup.

5.1 Experimental Setup

Model Configurations. We evaluate two settings: CycleGen-8B, built on Qwen3-8B and trained with 21k samples synthesized by our Phase I Data Engine, and CycleGen-Agent-32B, using Qwen3-32B as the backbone with our cycle graph construction and bi-directional refinement mechanisms.

Training Details. All experiments are conducted on 8 $\times$ NVIDIA H200 GPUs (141GB each). For SFT, we set batch size to 16 (with gradient accumulation), learning rate to 1×10^{-4}, and train for 3 epochs. The SFT training set size is 15k generated by CycleGen, with a test set of 1k and a validation set of 1k. The RL preference dataset size is 6k. For RL alignment, we use learning rate 1×10^{-6} for 2 epochs.

Benchmarks and Metrics. We adopt three benchmarks to comprehensively assess different aspects of long-form text generation.

- **WritingBench** (Wu et al., 2025c) evaluates writing quality across 6 domains (Academic & Engineering, Finance & Business, Politics & Law, Literature & Art, Education, Advertising & Marketing) and 3 requirements (Style, Format, Length) using a critic model on a 1–10 scale. This benchmark focuses on literary expressiveness, logical coherence, and instruction adherence in one-shot generation scenarios.
- **LongGenBench** (Wu et al., 2025b) tests agent planning capabilities through extremely long generation tasks (up to 16k tokens) under strict structural constraints. It measures five metrics: Completion Rate (whether all subtasks are completed sequentially), and instruction adherence across single-shot (Acc. Once), range-based (Acc. Range), and periodic (Acc. Periodic) constraints, with Average Accuracy (Avg. Acc.) summarizing overall performance.
- **CycleBench** is our proposed benchmark consisting of 100 long-form writing tasks (50 Chinese, 50 English), each involving 2–4 cyclic logical dependencies. Unlike existing benchmarks, CycleBench specifically evaluates whether models can resolve circular constraints where future content retroactively constrains earlier passages, assessing both global planning ability and logical self-consistency. We evaluate model performance using metrics such as Pass Rate, Score thresholds (Score $\geq$ 6, Score $\geq$ 8), Length Satisfaction Rate (LS), and Length Deviation (LD).

5.2 Main Results

Results on WritingBench. We evaluate one-shot generation performance on WritingBench, Table 1 shows the results. CycleGen-8B achieves an average score of 7.57, substantially outperforming similar-scale baselines including LongWriter-7B, Qwen-2.5-7B-filtered, and Qwen3-8B. Notably, CycleGen-8B even surpasses the larger Qwen3-32B at 7.48, demonstrating that our Data Engine substantially enhances long-form text generation capabilities in smaller models.

For the agent framework, CycleGen-Agent-32B achieves 7.76, outperforming the reasoning-enhanced Deepseek-R1 at 7.70 and ranking second only to Claude-3.7-thinking at 7.91. This further validates the effectiveness of graph-structured guidance in raising the upper bound of long-form writing quality.

Results on LongGenBench. We evaluate agent planning capabilities on LongGenBench, which requires generating extremely long texts under strict structural constraints. Table 2 shows the results.

CycleGen-Agent-32B achieves competitive performance across all metrics. On range-based accuracy, CycleGen-Agent-32B reaches 0.86, sub-

Model	Avg	Languages		Domains						Requirements					
		ZH	EN	D1	D2	D3	D4	D5	D6	R1	C	R2	C	R3	C
Proprietary LLMs															
Claude-3.7-thinking	**7.91**	7.9	**7.9**	**7.9**	7.8	**7.8**	**8.0**	**8.0**	**8.1**	**7.9**	**8.7**	**8.0**	8.4	**8.0**	**8.1**
Qwen-Max	7.16	7.2	7.1	7.1	6.9	7.0	7.3	7.4	7.5	7.2	8.3	7.3	7.8	7.2	7.5
GPT-4o	6.81	6.9	6.7	6.8	6.6	6.7	6.8	7.0	7.1	6.9	8.0	7.0	7.5	6.8	6.8
Open-source LLMs															
Deepseek-R1 (Guo et al., 2025)	7.70	**8.0**	7.5	7.6	7.4	7.6	7.8	7.8	**8.1**	7.7	8.4	7.9	8.3	7.7	7.5
Deepseek-V3 (DeepSeek-AI et al., 2025)	6.35	6.3	6.4	6.4	6.1	6.2	6.3	6.6	6.8	6.4	7.6	6.5	7.1	6.5	6.4
Qwen3-32B (Yang et al., 2025)	7.48	7.7	7.2	7.4	7.3	7.5	7.5	7.5	7.8	7.6	8.2	7.5	8.3	7.5	7.3
Qwen3-8B (Yang et al., 2025)	6.65	6.8	6.5	6.7	6.7	6.6	6.4	6.9	6.8	6.7	7.2	6.8	7.9	6.5	6.7
Qwen2.5-72B (Qwen et al., 2025)	6.53	6.5	6.5	6.5	6.3	6.5	6.5	6.7	6.7	6.7	7.2	6.7	7.8	6.8	6.9
Capability-enhanced LLMs															
LongWriter-7B (Bai et al., 2024)	6.27	6.2	6.4	6.4	6.4	6.3	6.0	6.5	6.0	6.3	7.4	6.3	6.7	6.3	6.8
Qwen-2.5-7B-filtered (Wu et al., 2025c)	7.44	7.7	7.2	7.4	7.2	7.5	7.3	7.7	7.7	7.5	8.4	7.6	8.1	7.4	7.2
CycleGen-8B(Ours)	7.57	7.4	7.4	7.5	7.9	7.4	7.3	7.8	7.8	7.7	8.0	7.5	**8.5**	7.4	7.5
Agent Framework															
CycleGen-Agent-32B(Ours)	7.76	7.8	7.7	**7.9**	**8.0**	7.7	7.5	7.8	7.7	7.7	7.9	7.8	7.7	7.4	6.4

Table 1: WritingBench performance of different LLMs across 6 domains and 3 writing requirements evaluated with our critic model (scale: 1-10). The six domains include: (D1) Academic & Engineering, (D2) Finance & Business, (D3) Politics & Law, (D4) Literature & Art, (D5) Education, and (D6) Advertising & Marketing. The three writing requirements assessed are: (R1) Style, (R2) Format, and (R3) Length. Here, "C" indicates category-specific score.

Model / Agent Framework	Comp. Rate	Acc. Once	Acc. Range	Acc. Periodic	Avg. Acc.
Llama-3.1-70B-Instruct (Touvron et al., 2023)	0.79	0.50	0.51	0.18	0.39
GPT-4o	0.63	0.47	0.45	0.20	0.37
Mixtral-8x7B-Instruct-v0.1	0.83	0.42	0.45	0.24	0.37
Qwen2-7B-Instruct	0.40	0.48	0.39	0.19	0.35
GPT-4o-mini	0.97	0.42	0.34	0.18	0.31
Qwen3-8B	0.27	0.15	0.09	0.10	0.11
Llama-3.3-70B-Instruct + CogWriter (Wan et al., 2025)	**1.00**	0.76	0.79	0.55	0.70
GPT-4o + CogWriter (Wan et al., 2025)	0.91	0.80	0.76	**0.67**	0.74
CycleGen-Agent-32B(Ours)	0.96	**0.85**	**0.86**	0.53	**0.75**

Table 2: Agent capability evaluation on LongGenBench. We focus on comparing agent frameworks. The results show that CycleGen achieves higher control precision and task success rates in long-form text generation compared to existing agent frameworks such as CogWriter.

stantially outperforming GPT-4o + CogWriter at 0.76, demonstrating superior length control through our structure-guided refinement mechanism. For single-shot instruction adherence, CycleGen-Agent-32B achieves 0.85 compared to 0.76 for Llama-3.3-70B + CogWriter, validating the effectiveness of our progressive alignment strategy. Overall, CycleGen-Agent-32B attains an average accuracy of 0.75, exceeding existing agent frameworks. Overall, these results indicate that graph-based planning with Loop-Aware Refinement provides more precise control than conventional planning approaches.

Results on CycleBench. We further evaluate model performance under highly complex constraints on CycleBench. Table 3 presents the results.

CycleGen-8B demonstrates strong long-range logical consistency, achieving a Pass Rate of 0.50, compared to 0.34 for the base Qwen3-8B and 0.46 for the larger Qwen3-32B. It also attains a higher high-quality generation rate (Score $\geq$ 8) of 0.56. These results suggest that internalizing graph-based planning into model parameters substantially improves long-form text generation.

We further analyze agent-based variants. CycleGen-Agent-32B (Refine) improves the Pass Rate from 0.58 to 0.65, validating the effectiveness of Loop-Aware Refinement in repairing logical inconsistencies and strengthening cyclic closure in generated texts.

5.3 Ablation Study

To quantify the contributions of different stages in the CycleGen framework, we conducted an ablation study. The results are presented in Table 4.

Effect of Progressive Alignment. Comparing Rows (1) to (4), the impact of the three-stage alignment strategy can be observed:

- **Stage 1 (GC-IPO):** Introducing logical con-

Model	Pass Rate	Score ≥ 6	Score ≥ 8	LS(%)	LD(words)	Len.
LongWriter-7B	0.27	0.34	0.09	0.10	7,565	0.61
Qwen-2.5-7B-filtered	0.22	0.35	0.21	0.01	12,823	0.09
Qwen3-8B	0.34	0.38	0.20	0.17	6,388	0.70
Qwen3-32B	0.46	0.56	0.32	0.07	6,914	0.58
CycleGen-8B	0.50	0.84	0.56	0.11	6,581	0.59
Agent Framework						
CycleGen-Agent-32B (Raw)	0.58	0.97	0.68	0.23	3,879	0.83
CycleGen-Agent-32B (Refine)	**0.65**	**0.99**	**0.74**	**0.21**	**3,933**	**0.84**

Table 3: Evaluation results on CycleBench. **LS** denotes the average length satisfaction rate, and **LD** represents the average length deviation (in word count).

	Configuration	Pass Rate	Score ≥ 6	Score ≥ 8	LS(%)	LD(words)
(1)	CycleGenSFT	0.51	0.79	0.49	0.09	6,681
(2)	+ Stage 1	0.53	**0.86**	0.42	0.17	6,547
(3)	+ Stage 2	0.48	0.81	0.53	0.11	**6,472**
(4)	+ Stage 3	0.50	0.84	**0.56**	0.11	6,581
(5)	MixDPO	0.55	0.70	0.46	0.18	6,873
(6)	CycleGen (Raw)	0.58	0.97	0.68	**0.23**	**3,879**
(7)	CycleGen (Refine)	**0.65**	**0.99**	**0.74**	0.21	3,933

Table 4: Ablation study analysis. We compare the contributions of each stage in the progressive training (Rows 2–4) and contrast them with the mixed training strategy (Row 5). LS denotes the average length satisfaction rate, and LD denotes the average length deviation in word count.

sistency constraints (Row 2) improves the overall pass rate (Score $\geq$ 6) from 0.79 to **0.86**, indicating enhanced local coherence in generated texts.

- **Stage 2 (DPO):** Adding strict length constraints (Row 3) reduces the average length deviation rate (**LS**) from 0.17 to **0.11**, while the average length deviation (**LD**) decreases to **6,472 words**, demonstrating more precise control over text length.

- **Stage 3 (SimPO):** The final stage (Row 4), which optimizes quality preferences with length normalization, maintains LS at 0.11 while increasing high-quality output (Score $\geq$ 8) to **0.56**, achieving a balanced trade-off among logical consistency, length, and quality.

Comparison with Mixed DPO Training. Comparing the progressive alignment strategy with MixDPO (Row 5), where all data are trained in a single-stage DPO, we observe that MixDPO achieves moderate pass rates but underperforms in high-quality generation (Score $\geq$ 8) and length control (LS 0.18 vs 0.11). This indicates that decoupling multi-objective optimization into stages is more effective than end-to-end mixed training.

Impact of the Refinement Mechanism. In the Agent setting (Rows 6 and 7), incorporating the Refinement mechanism increases the Pass Rate by 7 percentage points and improves high-quality scores. This confirms that explicit loop-aware revision is valuable for handling complex cyclic dependencies in long-text generation.

6 Conclusion

This work tackles the fundamental tension between the inherently linear decoding process of LLMs and the circular logical dependencies common in long-form text generation. We introduce CycleGen, a framework that explicitly models backward dependencies via a cycle graph data engine, and employs a progressive alignment strategy to mitigate multi-objective optimization challenges. To comprehensively evaluate structured reasoning and coherence in long narratives, we also release CycleBench, a benchmark for assessing bidirectional logical consistency in long-form texts. Empirical results demonstrate that CycleGen substantially improves structural control and logical coherence even in small parameter models, highlighting the effectiveness of graph-guided generation and bidirectional refinement for long-text writing.

Limitations

Despite its effectiveness, this study has several limitations. First, while constructing a directed acyclic graph enables maximal parallelism of logical dependencies, the LLM-driven verification and iterative refinement steps incur substantial computational overhead. Second, the current graph modeling is restricted to chapter-level granularity, leaving finer-grained intra-chapter dependencies largely overlooked. Finally, the framework's performance is limited by the teacher model's reasoning capabilities. Mistakes in causal relationship identification during graph construction can propagate into the synthesized training data. Future work will aim to improve synthesis efficiency and extend graph modeling to finer-grained textual hierarchies, further enhancing the quality and consistency of long-form text generation.

References

Ayomide Akinfaderin, Sriram Subramanian, and Anurag Sehwag. 2025. Plan-and-write: Structure-guided length control for llms without model retraining. In *First International KDD Workshop on Prompt Optimization*.

Cem Anil, Yuhuai Wu, Anders Andreassen, Bo Dai, Yanzhi Li, Jure Leskovec, Daniel Johnson, and 1 others. 2022. Exploring length generalization in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 38546–38556.

Y. Bai, J. Zhang, X. Lv, and 1 others. 2024. Longwriter: Unleashing 10,000+ word generation from long-context llms. *arXiv preprint arXiv:2408.07055*.

Jonas Becker, Jan Philip Wahle, Bela Gipp, and Terry Ruas. 2024. Text generation: A systematic literature review of tasks, evaluation, and challenges. *arXiv preprint arXiv:2405.15604*.

Benjamin Butcher, Michael O'Keefe, and James Titchener. 2025. Precise length control for large language models. *Natural Language Processing Journal*, 11:100143.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Zihan Feng, Hao Zhou, Kai Mao, Yuhui Li, and Jie Zhang. 2024. Freectrl: Constructing control centers with feedforward layers for learning-free controllable text generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7627–7640.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024. A general theoretical paradigm to understand learning from human preferences. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 4447–4455. PMLR.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*.

Jian Guan, Xiaoxi Mao, Changjie Fan, Zitao Liu, Wenbiao Ding, and Minlie Huang. 2021. Long text generation by modeling sentence-level and discourse-level coherence. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6379–6393, Online. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Nelson F. Liu, Kevin Lin, John Hewitt, Bhargav Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, volume 37, pages 124198–124235. Curran Associates, Inc.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741. Curran Associates, Inc.

J. Su, M. Ahmed, Y. Lu, and 1 others. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Kaiyang Wan, Honglin Mu, Rui Hao, Haoran Luo, Tianle Gu, and Xiuying Chen. 2025. A cognitive writing perspective for constrained long-form text generation. *arXiv preprint arXiv:2502.12568*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Yuhan Wu, Yuxin Bai, Zhen Hu, Jie Zhang, and Yuhui Li. 2025a. Superwriter: Reflection-driven long-form generation with large language models. *arXiv preprint arXiv:2506.04180*.

Yuhan Wu, Min Seok Hee, Zhen Hu, Yuxin Bai, Jie Zhang, and Yuhui Li. 2025b. Longgenbench: Benchmarking long-form generation in long context llms. In *The Thirteenth International Conference on Learning Representations*.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. 2025c. Writingbench: A comprehensive benchmark for generative writing. *Preprint*, arXiv:2503.05244.

Haotian Xia, Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2025. Storywriter: A multi-agent framework for long story generation. *arXiv preprint arXiv:2506.16445*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Kevin Yang, Dan Klein, Nanyun Peng, and Yuandong Tian. 2023. DOC: Improving long story coherence with detailed outline control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3378–3465, Toronto, Canada. Association for Computational Linguistics.

Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. 2022. Re3: Generating longer stories with recursive reprompting and revision. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4393–4479, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Peng Yuan, Chuan Tan, Shuo Feng, Yuhui Li, and Jie Zhang. 2025. From sub-ability diagnosis to human-aligned generation: Bridging the gap for text length control via markergen. *arXiv preprint arXiv:2502.13544*.

A Appendix

A.1 CycleGen Data Synthesis Pipeline

Algorithm 1 CycleGen Data Synthesis Pipeline

Require: User instruction I, structured outline $\mathcal{O} = \{v_1, \ldots, v_N\}$, similarity threshold τ
Ensure: Preference pair $(\mathcal{Y}_{raw}, \mathcal{Y}_{refined})$

$V \leftarrow \mathcal{O}, E \leftarrow \emptyset$
$\mathbf{E} \leftarrow \textsc{Embedding}(\mathcal{O})$
for all $(v_i, v_j) \in V \times V$ **do**
  $s_{ij} \leftarrow \cos(\mathbf{e}_i, \mathbf{e}_j)$
  if $s_{ij} > \tau$ **then**
    $(u, v, w) \leftarrow \text{LLM-}\textsc{Verify}(v_i, v_j)$
    if dependency exists **then**
      $E \leftarrow E \cup \{(u \rightarrow v, w)\}$
    end if
  end if
end for
$G \leftarrow (V, E)$
$E_{cut} \leftarrow \textsc{MinFAS}(G)$
$G_{DAG} \leftarrow (V, E \setminus E_{cut})$
Initialize array $\mathcal{Y}_{raw}$ of size N
$Batches \leftarrow \textsc{LayeredTopologicalSort}(G_{DAG})$
for all $batch \in Batches$ **do**
  parallel for $v_i \in batch$ **do**
  $C_i \leftarrow \textsc{Retriever}(\text{Predecessors}_{G_{DAG}}(v_i), \mathcal{Y}_{raw})$
  $y_i \leftarrow \text{LLM-}\textsc{Gen}(I, v_i, C_i)$
  $\mathcal{Y}_{raw}[i] \leftarrow y_i$
  end parallel
end for
$\mathcal{Y}_{refined} \leftarrow \mathcal{Y}_{raw}$
for all $(v_j \rightarrow v_i) \in E_{cut}$ **do**
  $(y_i', y_j') \leftarrow \text{LLM-}\textsc{Refine}(\mathcal{Y}_{refined}[i], \mathcal{Y}_{refined}[j])$
  $\mathcal{Y}_{refined}[i] \leftarrow y_i'$
  $\mathcal{Y}_{refined}[j] \leftarrow y_j'$
end for
return $(\mathcal{Y}_{raw}, \mathcal{Y}_{refined})$

Algorithm 1 summarizes how CycleGen constructs preference pairs by detecting logical dependencies in an outline, enabling parallel DAG-based generation, and subsequently refining broken cyclic relations to restore global coherence.

A.2 Prompt Design Details

Outline

```
You are a professional structure planning and content architecture expert
- User requirement: {query}
- Total word count limit: {word_count}
- Style: {style}
1. Output format:
  - Output only the outline in Markdown.
  - Do not include explanations or any non-outline content.
  - Each heading must be immediately followed by a paragraph summary on
a new line, enclosed in parentheses.
  - The main title is not part of the heading hierarchy.
2. Structure:
  - Determine the number of level-1 headings dynamically according to
word count and style; each should form a complete logical unit.
  - Use only the necessary heading depth to clearly express the logic.
  - Ensure the structure reasonably fits within the total word count.
3. Content:
  - Every heading must include a summary capturing its core logic.
  - The outline should be complete, clear, and minimally layered.
  - If no word count is specified, generate a reasonably sized outline.
  - Strictly use the language specified in the user’ s requirements.
```

Figure 4: Generate Outline prompt.

Figure 4 shows the outline-generation prompt, which guides the model to produce a coherent, multi-level global structure rather than detailed text.

Task

```
You are a senior content architect and writing task planning expert.
Rules:
The output must be a JSON list, where each element corresponds to one level-1
heading task block of the article.
Each task block must strictly contain the following fields (and be filled according
to the meanings below):
  - "article_title": the overall title of the article
  - "title": the level-1 heading corresponding to this task block; it may be empty
  - "subtitles": the list of subheadings under this level-1 heading
  - "length_limit": the word-count limit for this section
  - "style": the writing style requirements
  - "notes": special instructions or additional requirements for writing this
section
prompts, or wrapper text.
The input information has the following structure:
  - User requirement: {query}
  - Outline: {outline}
  - Total word count limit: {word_count}
  - Writing style: {style}
```

Figure 5: Generate Task prompt.

Figure 5 presents the task-generation prompt that maps the global outline into chapter-level writing objectives aligned with the overall plan.

Judge

```
You will determine the logical dependency relationship between two chapter
nodes:
Node A:
- Title: "{title_i}"
- Summary: "{summary_i}"
Node B:
- Title: "{title_j}"
- Summary: "{summary_j}"
Their known similarity is: {sim:.4f} (this can help with the judgment but is not
the only basis).
1. If the content of node A is the premise of node B, output "i_to_j".
2. If the content of node B is the premise of node A, output "j_to_i".
3. If the two are logically related and suitable for subsequent refinement
between the two chapters, output "both".
4. If there is no obvious logical dependency, output "none".
Requirements:
- Output strictly in JSON format:
{{"direction": "i_to_j/j_to_i/both/none", "reason": "brief explanation of the
logical relationship"}}
```

Figure 6: Judge prompt.

Figure 6 illustrates the judge prompt used to infer directed dependency or refinement relations between chapter plans for graph construction.

Generate

```
You are to generate the complete markdown body for this chapter based on
the following information:
- Outline: {outline}
- Writing style: {style}
- Summary of content already generated: {previous_contents}
- Chapter title: {title}
- Subheadings to include: {subtitles}
- Summary of this chapter: {notes}
- Chapter length: at least {length_limit} words, at most {length_limit} * 1.05
words

Requirements:
1. Output only the final complete markdown body of this chapter "{title}",
and do not output any additional content.
2. Strictly organize the content according to the length requirement of about
{length_limit} words and the specified writing style: {style}.
```

Figure 7: Generate Chapter prompt.

Figures 7 and 8 depict the chapter generation and refinement prompts, which respectively produce draft content conditioned on predecessors and revise cyclically related chapters to improve global coherence.

Refine

You will perform precise relationship refinement on two chapter texts so that their logical dependency and reference relationships become clearer.
The input includes:
Relationship direction: {direction}; Reason for judgment: {reason}
Chapter A (id={i}): {text_i}; Chapter B (id={j}): {text_j}
Higher-level context (for alignment and continuity reference only; must not be copied verbatim):
Parent of A: {prev_i}; Parent of B: {prev_j}
Task requirements:
Without changing the original meaning, supplement connecting sentences, premise statements, and relational descriptions so that A and B are logically consistent and mutually readable.When the direction is j_to_i, prioritize making B's role as support/foundation/premise for A explicit; when it is both, emphasize bidirectional reference and mutual corroboration.
Nothing may be output outside the JSON. Strictly generate in the language provided by the user.
Output example (format illustration):
{{"A_new": "...", "B_new": "..."}}

Figure 8: Refine Chapter prompt.

A.3 Additional Experimental Analysis

As illustrated in Figure 9, CycleGen achieves consistent improvements over baseline models across almost evaluation dimensions on WritingBench, indicating the robustness of the proposed framework.

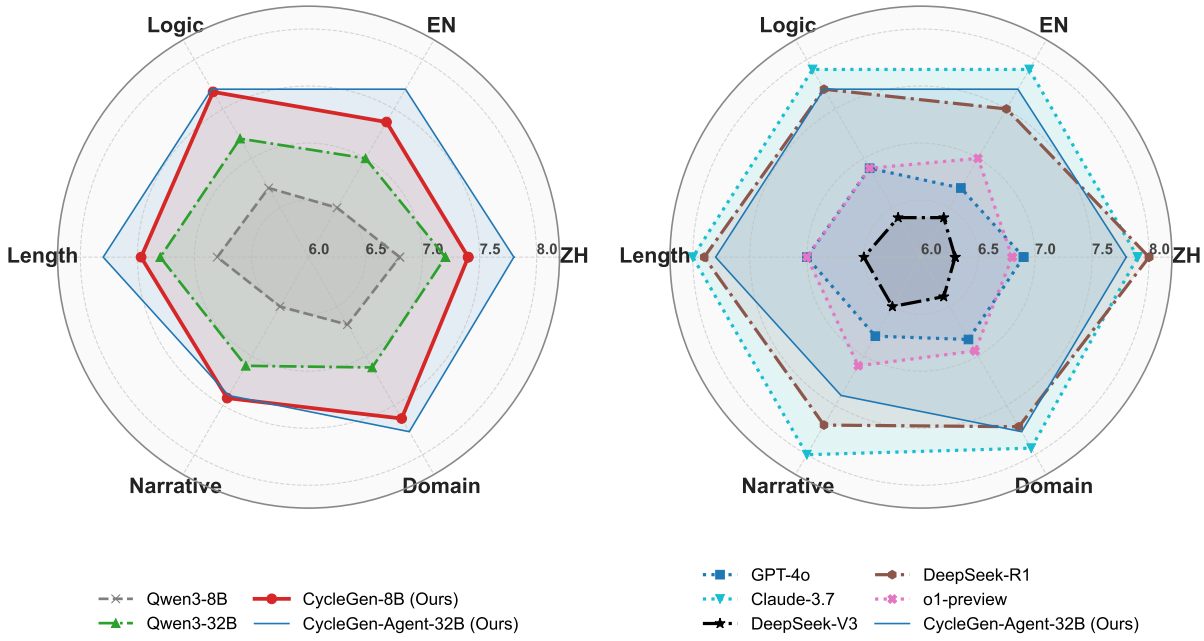

Figure 9: Radar chart comparison of CycleGen models across different dimensions on WritingBench. The left panel compares against open-source Qwen models, while the right panel compares against closed-source SOTA models.